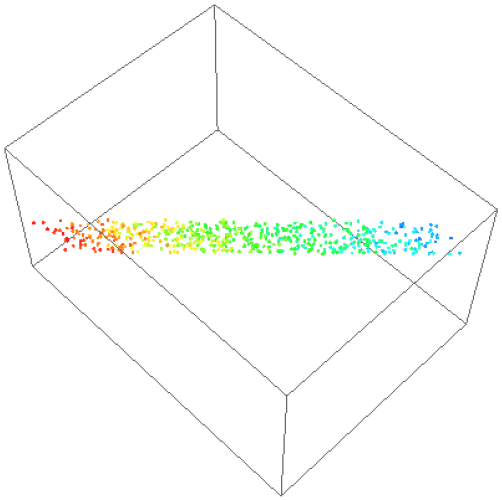
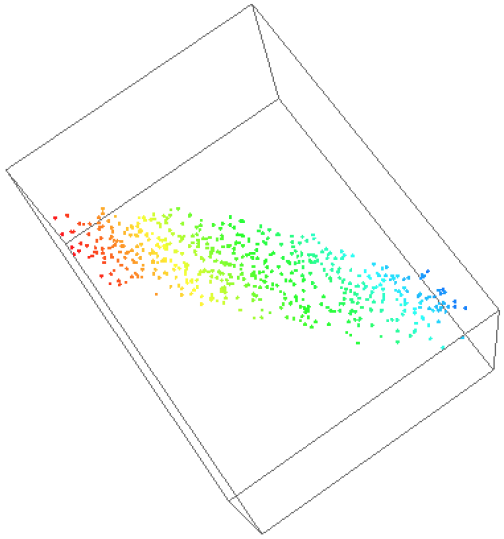
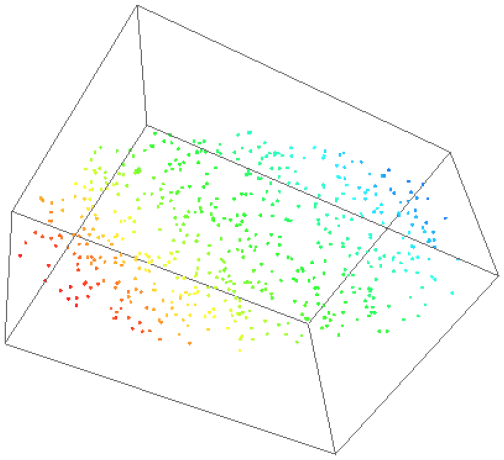
Principal Component Analysis

Why PCA?

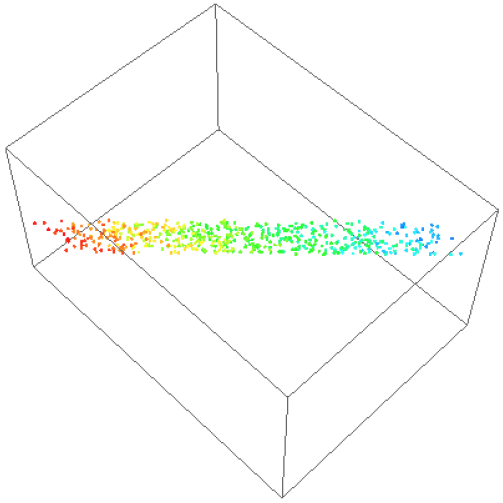
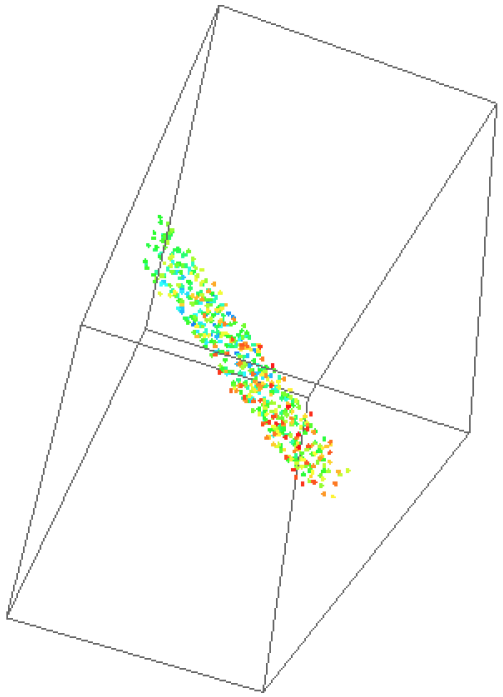
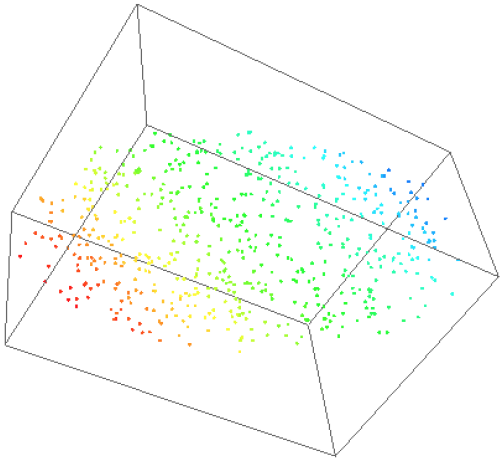
- Often, high dimensional features are correlated
- PCA is a way to do dimention reduction
- It projects the high dimensional data into lower dimension, such that the variance is retained as much as possible
- It is a great way to visualize high dimensional data

- Why dimension reduction: some high D data has actual intrinsic low dimension

view 1	view 2	view 3
		

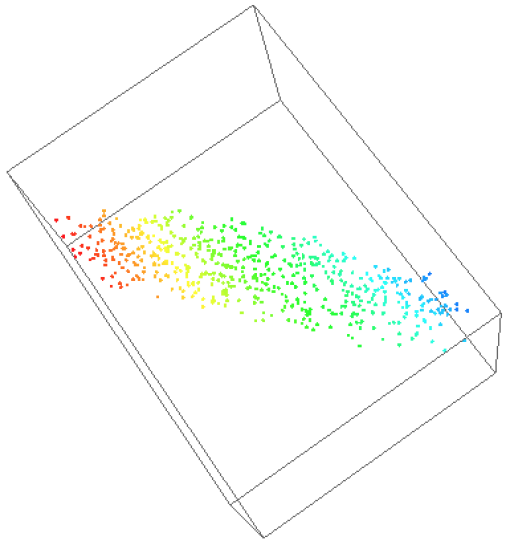
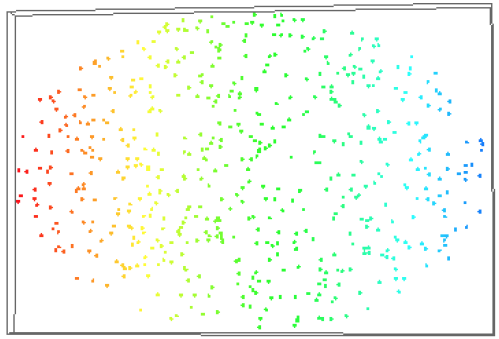
low variation	even lower	highest variation

- We can reduce dimension by projection
- Best projected data: max variation

low variation	even lower	highest variation
		

3D	2D

- Projecting to lower dimension: from high D, to lower D with highest variation

3D	2D
	

A Small Example

- consumption in grams (per person, per week) of 17 different types of foodstuff measured and averaged in the four countries of the United Kingdom in 1997, from http://sebastianraschka.com/Articles/2015_pca_in_3_steps.html (http://sebastianraschka.com/Articles/2015_pca_in_3_steps.html)
- How does the countries relate to each other?
- Are some countries closer in this food preference than others?

In [1]:
```
food = pd.DataFrame({"country":["England","Wales","Scotland","NIreland"],
"Cheese":[105.,103,103,66],
"CarcassMeat":[245,227,242,267],
"OtherMeat":[685,803,750,586],
"Fish":[147,160,122,93],
"FatsAndOils":[193,235,184,209],
"Sugars":[156,175,147,139],
"FreshPotatoes":[720,874,566,1033],
"FreshVeg":[253,265,171,143],
"OtherVeg":[488,570,418,355],
"ProcessedPotatoes":[198,203,220,187],
"ProcessedVeg":[360,365,337,334],
"FreshFruit":[1102,1137,957,674],
"Cereals":[1472,1582,1462,1494],
"Beverages":[57,73,53,47],
"SoftDrinks":[1374,1256,1572,1506],
"AlcoholicDrinks":[375,475,458,135],
"Confectionery":[54,64,62,41]})
mat = food.drop("country", 1).values
labels = food["country"].values
```

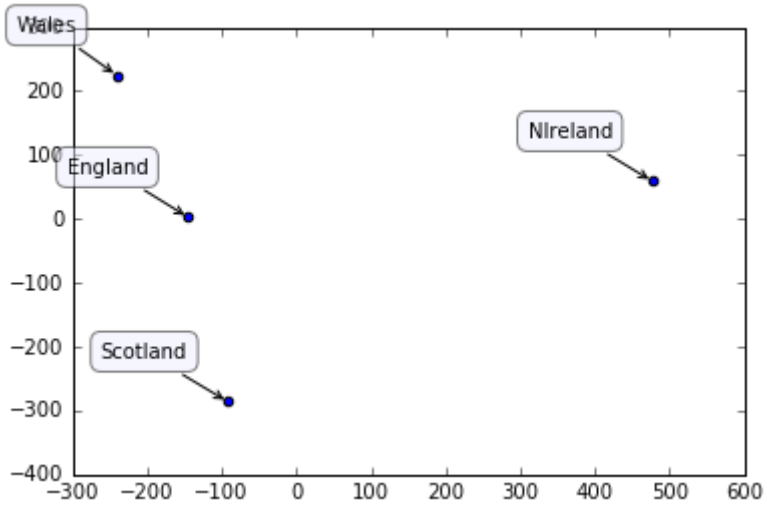
In [2]:
```
food
```

Out[2]:

	AlcoholicDrinks	Beverages	CarcassMeat	Cereals	Cheese	Confectionery	FatsAndOils	Fish	FreshFruit	FreshPotatoes
0	375	57	245	1472	105.0	54	193	147	1102	720
1	475	73	227	1582	103.0	64	235	160	1137	874
2	458	53	242	1462	103.0	62	184	122	957	566
3	135	47	267	1494	66.0	41	209	93	674	1033

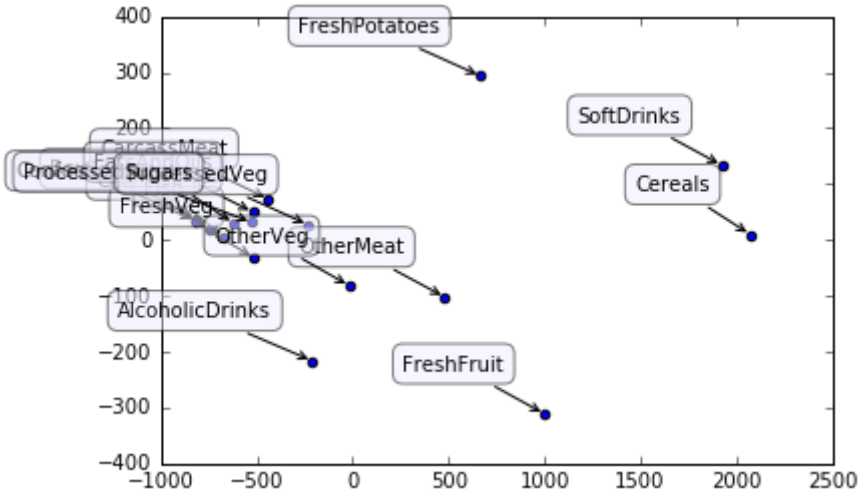
- Staring at the data does not help
- But with dimension reduction, we can visualize
- Seems that England & Scotland are closer

In [3]:
```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
mat_normalized = StandardScaler(with_std=False).fit_transform(mat) #try changing with_std

pca = PCA(n_components=2)
pca.fit(mat_normalized)
mat_pca = pca.fit_transform(mat_normalized)
plt.scatter(mat_pca[:,0],mat_pca[:,1])

for label, x, y in zip(labels, mat_pca[:, 0], mat_pca[:, 1]):
    plt.annotate(label, xy = (x, y), xytext = (-20, 20),
        textcoords = 'offset points', ha = 'right', va = 'bottom',
        bbox = dict(boxstyle = 'round,pad=0.5', fc = '#eeeeff', alpha = 0.5),
        arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad=0'))
```



- We can also visualize how foods relate to each other
- If two food items are consumed by similar countries, they are related

In [4]:
```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
mat_transposed = np.transpose(mat)
labels_transpose = food.columns.delete(-1)
mat_transposed_normalized = StandardScaler(with_std=False).fit_transform(mat_transposed)
pca.fit(mat_transposed_normalized)
mat_transpose_pca = pca.fit_transform(mat_transposed_normalized)
plt.scatter(mat_transpose_pca[:,0],mat_transpose_pca[:,1])
for label, x, y in zip(labels_transpose, mat_transpose_pca[:, 0], mat_transpose_pca[:, 1]):
    plt.annotate(label, xy = (x, y), xytext = (-20, 20),
        textcoords = 'offset points', ha = 'right', va = 'bottom',
        bbox = dict(boxstyle = 'round,pad=0.5', fc = '#eeeeff', alpha = 0.5),
        arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad=0'))
```



Derivation

- We have a feature matrix X of dimension $n \times k$.
- k is the number of features
- We want to find a projection matrix P (P is orthogonal) of dimension $k \times l$, so that, e.g.

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \\ x_{61} & x_{62} & x_{63} \end{pmatrix} \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \\ p_{31} & p_{32} \end{pmatrix} = \begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \\ z_{31} & z_{32} \\ z_{41} & z_{42} \\ z_{51} & z_{52} \\ z_{61} & z_{62} \end{pmatrix}$$

- The projection matrix P should maximize the co-variance of XP
- It is easier to consider one dimension at a time.
- So we are looking for a k-D **normalized** vector p so that vector Xp has the largest variance
- We assume that X is normalized, so that sum of each column of X is zero.
- This means that the sum of the vector $y = Xp$ is also zero: $e^T Xp = 0$, where e is the n-D vector of all 1's

- We want to maximize the same variance

$$var(Xp) = ((y - E(y))^T (y - E(y))/(n-1) = (y^T y)/(n-1) = p^T X^T Xp/(n-1)$$

- By linear algebra, we know that any symmetric matrix can be decomposed into
- $X^T X = U \Sigma U^T$ where U is $k \times k$ orthgogonal matrix.
- Σ is $k \times k$ diagonal matrix, with diagnal sorted from large to small $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k \geq 0$

- Each column of U is an eigenvector of $X^T X$
- Let $v = U^T p$ which is also a normalized vector ($v^T v = p^T U U^T p = 1$)
- Then we want to find v that maximize $v^T \Sigma v = \sum_{i=1}^{k} \lambda_i v_i^2$
- The maximum is aciieved when $v = e_1 = \{1, 0, 0, \ldots, 0\}$, and the variance is λ_1
- So $p = Uv = Ue_1 = u_1$, which is the largest eigenvector of $X^T X$

- In general, it can be proved that the best projection matrix that project from k to l-dimensional space is the matrix P consists of the first l-largest eigenvectors of X^TX
- The data after projection is XP, it is of $n \times l$ dimension. This is the end result of PCA.
- The sum of the sample variances of all columns of XP is $\lambda_1 + \lambda_2 + \ldots + \lambda_l$
- The portion of the variance it explains is

$$\frac{\lambda_1 + \lambda_2 + \ldots + \lambda_l}{\lambda_1 + \lambda_2 + \ldots + \lambda_l + \ldots \lambda_k}$$

- If $l = 2, 3$, we can visualize the result.
- E.g., consider the 17×4 matrix of Foods x countries.
- We want to reduce it to 17×2 so that we can see the 17 food in 2D visualization

In [5]:
```
X = mat_transposed_normalized
print("dimension of data = ", X.shape)
XTX = np.dot(X.T, X)
(sigma, U) = np.linalg.eig(XTX)
```

```
dimension of data =  (17, 4)
```

- The eigenvalues

In [6]:
```
idx = sigma.argsort()[::-1]
sigma = sigma[idx] # we sort so that largest eigens come first
U = U[:,idx]
sigma
```

Out[6]:
```
array([  1.35170683e+07,   2.79048225e+05,   1.25570483e+05,
         1.31058654e+04])
```

- The total sum of variances among the 4 features is the sum of the 4 eigenvalues, divided by $n - 1$

In [7]:
```
print("total sum var = ", sigma.sum()/(X.shape[0]-1))
X.shape
```

```
total sum var =  870924.551471
```

Out[7]: (17, 4)

- It is the same as the sum of the diagonal entries of the covarance matrix of the 4 feature vectors

In [8]:
```
np.trace(np.cov(X.T, ddof=1))
```

Out[8]: 870924.55147058831

- Using the first two eigenvectors would account for 97% of the variances

In [9]:
```
print("first two eigenv account for ", sigma[0:1].sum()/sigma.sum());
```

```
first two eigenv account for  0.970022907545
```

- Projection matrix using the first 2 eigenvectors

In [10]:
```
P = U[:, :2];
print("eigevectors = ",U, "P = ",P)
```

```
eigevectors =  [[ 0.49015722  0.28084203 -0.07019969  0.82215915]
 [ 0.49817035  0.34759699 -0.64010493 -0.47039166]
 [ 0.50423561  0.24821973  0.76259165 -0.32029268]
 [ 0.50726586 -0.85947138 -0.06157655 -0.01409337]] P =  [[ 0.49015722  0.28084203]
 [ 0.49817035  0.34759699]
 [ 0.50423561  0.24821973]
 [ 0.50726586 -0.85947138]]
```

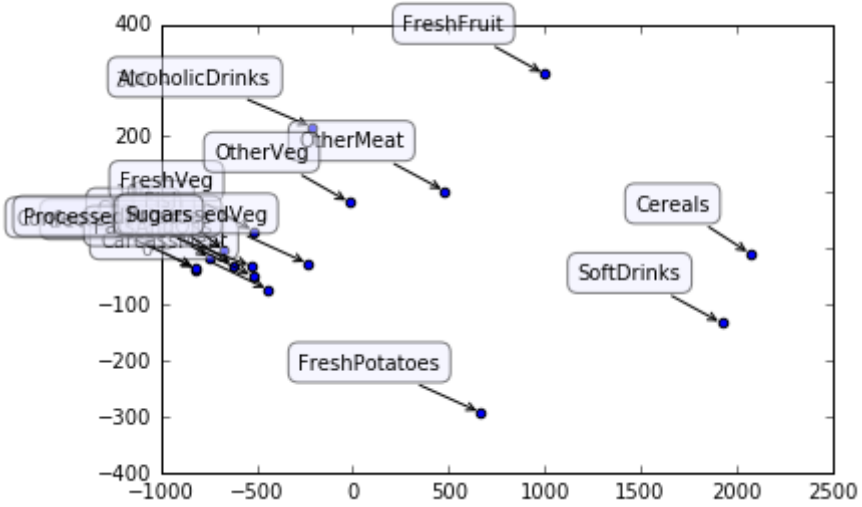
- After projection XP

In [11]:
```
pca_manual = np.dot(X, P); print("project data to 2D:", pca_manual)
```

```
project data to 2D: [[ -211.55027964   216.29791815]
 [ -816.53957374   -37.63934574]
 [ -440.77276332   -73.48128287]
 [ 2073.25362357    -9.63750641]
 [ -743.21708496   -17.64998849]
 [ -820.99905329   -34.21943892]
 [ -520.94266091   -49.85169653]
 [ -670.95811685    -4.53114723]
 [  999.61265884   311.18585113]
 [  666.30611977  -293.11795138]
 [ -516.62272677    30.9249895 ]
 [  479.81203414   102.22932813]
 [  -17.40726534    82.04228872]
 [ -527.44069324   -31.72630957]
 [ -233.76797909   -27.21977314]
 [ 1924.3677877   -133.48613099]
 [ -623.13402687   -30.11980439]]
```

- Visualizing the PCA (and notice that our manual method gives the same result as using `sklearn.decomposition.PCA`)

In [12]:
```
plt.scatter(pca_manual[:,0],pca_manual[:,1])
for label, x, y in zip(labels_transpose, pca_manual[:, 0], pca_manual[:, 1]):
    plt.annotate(label, xy = (x, y), xytext = (-20, 20),
        textcoords = 'offset points', ha = 'right', va = 'bottom',
        bbox = dict(boxstyle = 'round,pad=0.5', fc = '#eeeeff', alpha = 0.5),
        arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad=0'))
```



PCA recap

- The purpose is to reduce the dimension of the data: a $n \times k$ matrix X becomes a $n \times l$ matrix XP
- Normalize the data so that sum of values for each feature is 0
  - Alternatively we divide each feature by the sample standard deviation to make each feature value a z-value: we standardize
- We find the eigenvalues and vectors of $X^T X$
- If we are doing visualization, $l = 2$ or 3.
- Otherwise we select l so that a large percent of variation can be explains; P consists of the l eigenvectors

- Using sklearn to find the reduced dimension: we can see that even $l = 1$ gives us 97% of the variation

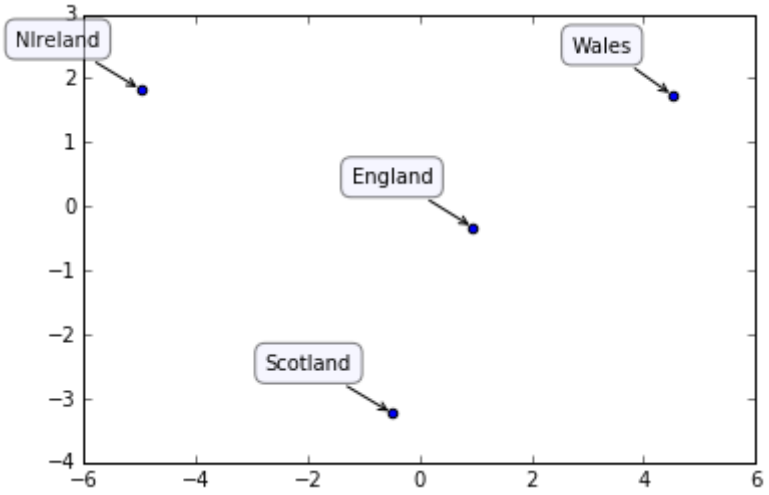
In [13]:
```
pca = PCA(n_components=4)
pca.fit(mat_transposed_normalized)
print(pca.explained_variance_ratio_)
```

```
[  9.70022908e-01   2.00252869e-02   9.01129168e-03   9.40513830e-04]
```

Should we standardize?

- The theorey only require that the sum of the feature values is zero, i.e., $X[:, i] = X[:, i] - \overline{X[:, i]}$
- Standardize means converting to z-value: $X[:, i] = (X[:, i] - \overline{X[:, i]})/s(X[:, i])$
- If the feature values are about the same scale, no need to divide by the sample standard deviation
- If not, consider scaling
- For the food case, standardization does not make a huge difference

In [14]:
```
mat_normalized = StandardScaler(with_std=True).fit_transform(mat) #scale with std!!
pca = PCA(n_components=2)
pca.fit(mat_normalized)
mat_pca = pca.fit_transform(mat_normalized)
plt.scatter(mat_pca[:,0],mat_pca[:,1])
for label, x, y in zip(labels, mat_pca[:, 0], mat_pca[:, 1]):
    plt.annotate(label, xy = (x, y), xytext = (-20, 20),
        textcoords = 'offset points', ha = 'right', va = 'bottom',
        bbox = dict(boxstyle = 'round,pad=0.5', fc = '#eeeeff', alpha = 0.5),
        arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad=0'))
```



How much is lost by PCA?

- By PCA and projecting to a lower dimension, we've lost some information
- The amount loss is the variance no explained by the PCA
- Which is the sum of the remaining eigenvalues $k - l$ eigenvalues not included
- We can visualize this effect using this example
- This is the famous "lenna" pic, a matrix of 512×512

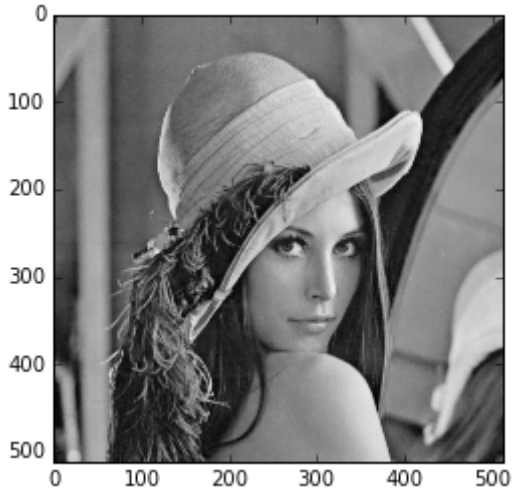
In [15]:
```
lenna = pd.read_csv("data/lenna.csv", header=None)
lenna[:2]
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	...	502	503	504	505	506	507	508	509
0	0.638	0.638	0.644	0.637	0.633	0.616	0.639	0.634	0.646	0.627	...	0.494	0.501	0.552	0.646	0.656	0.659	0.661	0.654
1	0.638	0.638	0.644	0.637	0.633	0.616	0.639	0.634	0.646	0.627	...	0.494	0.501	0.552	0.646	0.656	0.659	0.661	0.654

2 rows × 512 columns

In [16]:
```
plt.imshow(lenna)
plt.gray()
plt.show()
```



- This is a $512x512$ matrix. We now reduce it to $512 \times l$
- We are not standardize since pixel values are between 0 to 1 anyway
- PCA with $l = 20$ can explain 89% of the variance

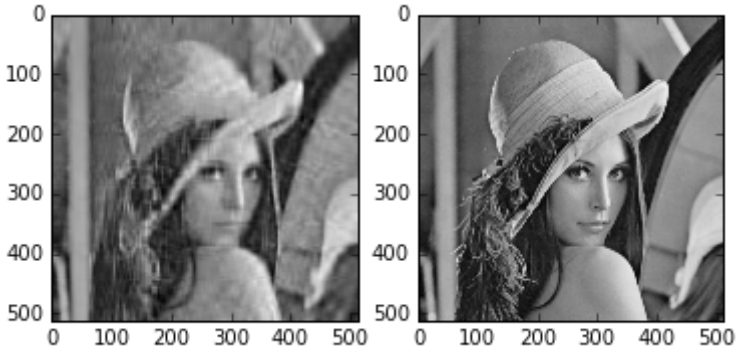
In [17]:
```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
lenna2 = StandardScaler(with_std=False).fit_transform(lenna) #not standardizing to avoid messing with
 the pic

pca = PCA(n_components=20)
pca.fit(lenna2)
lenna_pca = pca.fit_transform(lenna2)
pca.explained_variance_ratio_.sum()
```

Out[17]: 0.89020443967053808

- Now we project back from 20 to 512 dimensions
- This is done by multiply with P^T, i.e., $(XP)P^T$. The part in the bracket is the PCA of X.
- In the extreme case $P = U$, so $(XP)P^T = X$
- The new matrix has negative entries due to normalization
- We need to add back the mean of each column

In [18]:
```
mean = np.array([np.mean(x) for x in lenna.T.values])
f, (fig1, fig2) = plt.subplots(1, 2, sharey=False)
lenna_reborn = (pca.inverse_transform(lenna_pca) + mean)
fig1.imshow(lenna_reborn)
fig2.imshow(lenna)
print("origonal matrix: ", lenna.shape,", PCA: ", lenna_pca.shape)
plt.gray()
plt.show()
```

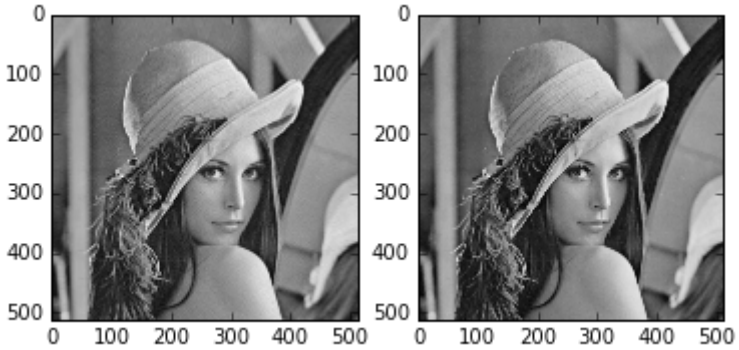
```
origonal matrix:  (512, 512) , PCA:  (512, 20)
```



- We can see at $l = 2$, we lost some details. Let's try $l = 100$. With that the PCA explained 98.9% of the variance
- The quality is very good. But we can now store the image of size about 1/5-th of the original!

In [19]:
```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
lenna2 = StandardScaler(with_std=False).fit_transform(lenna) #not standardizing to avoid messing with
 the pic
pca = PCA(n_components=100); pca.fit(lenna2)
lenna_pca = pca.fit_transform(lenna2)
print("variance explained = ",pca.explained_variance_ratio_.sum())
f, (fig1, fig2) = plt.subplots(1, 2, sharey=False)
lenna_reborn = (pca.inverse_transform(lenna_pca) + mean)
fig1.imshow(lenna_reborn); fig2.imshow(lenna)
plt.gray(); plt.show()
```

```
variance explained =  0.988918948684
```



Example: breakfast cereal

- Nutrition info
- from http://lib.stat.cmu.edu/DASL/Datafiles/Cereals.html (http://lib.stat.cmu.edu/DASL/Datafiles/Cereals.html)

In [20]:
```
cereal = pd.read_csv( data/cereal.csv", sep=",")
names = cereal["name"].values
cereal = cereal.drop(["name","mfr","type"],1)
cereal[:6] # we can see that there are missing values! (-1)
```

Out[20]:

	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
0	70	4	1	130	10.0	5.0	6	280	25	3	1.0	0.33	68.402973
1	120	3	5	15	2.0	8.0	8	135	0	3	1.0	1.00	33.983679
2	70	4	1	260	9.0	7.0	5	320	25	3	1.0	0.33	59.425505
3	50	4	0	140	14.0	8.0	0	330	25	3	1.0	0.50	93.704912
4	110	2	2	200	1.0	14.0	8	-1	25	3	1.0	0.75	34.384843
5	110	2	2	180	1.5	10.5	10	70	25	1	1.0	0.75	29.509541

- We replace missing values with the mean

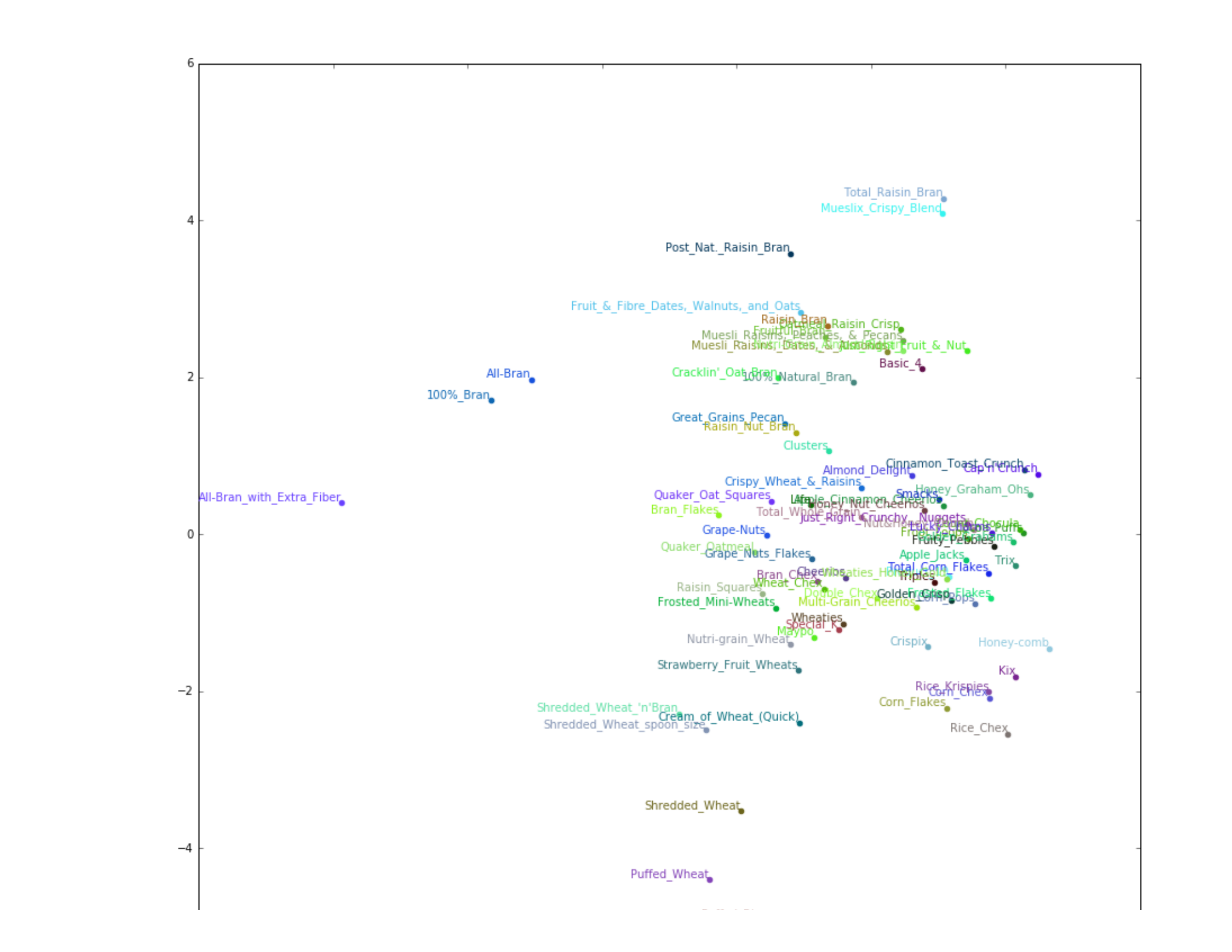
In [21]:
```
for vals in cereal.columns:
    c = cereal[vals]
    avg = np.mean(c[c != -1])
    cereal[vals] = c.replace(-1, avg)
cereal[:6]
```

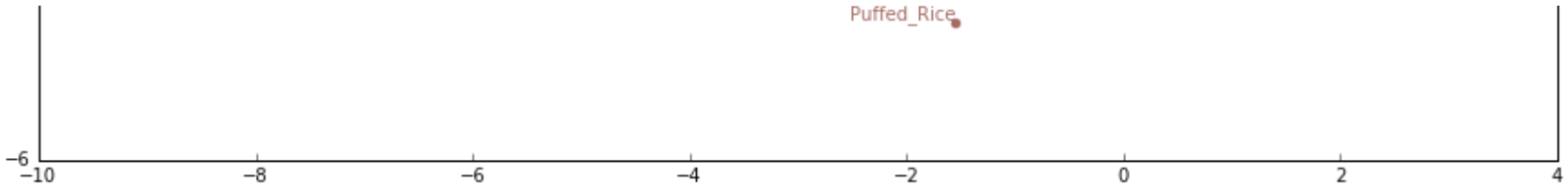
Out[21]:

	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
0	70	4	1	130	10.0	5.0	6.0	280.000000	25	3	1.0	0.33	68.402973
1	120	3	5	15	2.0	8.0	8.0	135.000000	0	3	1.0	1.00	33.983679
2	70	4	1	260	9.0	7.0	5.0	320.000000	25	3	1.0	0.33	59.425505
3	50	4	0	140	14.0	8.0	0.0	330.000000	25	3	1.0	0.50	93.704912
4	110	2	2	200	1.0	14.0	8.0	98.666667	25	3	1.0	0.75	34.384843
5	110	2	2	180	1.5	10.5	10.0	70.000000	25	1	1.0	0.75	29.509541

In [22]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import random
#generate 77 random colors, one for each cereal
random.seed(123)
color = ["#%06x" % random.randint(0, 0xAAAAAA) for i in range(0, cereal.shape[0])]

from pylab import rcParams #set figure size
rcParams['figure.figsize'] = 15, 15
```

In [23]:
```
cereal2 = StandardScaler(with_std=True).fit_transform(cereal) #standardize
pca = PCA(n_components=2) #get PCA
pca.fit(cereal2)
cereal_pca = pca.fit_transform(cereal2)
#scatter plot
for x, y, c in zip(cereal_pca[:,0], cereal_pca[:,1], color):
    plt.scatter(x,y,color=c)
#labels
for label, x, y, c in zip(names, cereal_pca[:,0],cereal_pca[:,1], color):
    plt.annotate(label, xy = (x, y), xytext = (-0, 0),
        textcoords = 'offset points', ha = 'right', va = 'bottom', color=c)
```


6
4
2
0
−2
−4
Total_Raisin_Bran
Mueslix_Crispy_Blend
Post_Nat._Raisin_Bran
Fruit_&_Fibre_Dates,_Walnuts,_and_Oats
Raisin_Bran
Raisin_Crisp
Muesli_Raisins,_Peaches,_&_Pecans
Fruit_&_Nut
Basic_4
All-Bran
Cracklin'_Oat_Bran
100%_Natural_Bran
100%_Bran
Great_Grains_Pecan
Raisin_Nut_Bran
Clusters
Cinnamon_Toast_Crunch
Almond_Delight
Cap'n'Crunch
Crispy_Wheat_&_Raisins
Honey_Graham_Ohs
All-Bran_with_Extra_Fiber
Quaker_Oat_Squares
Smacks
Honey_Nut_Cheerios
Bran_Flakes
Just_Right_Crunchy__Nuggets
Grape-Nuts
Fruity_Pebbles
Quaker_Oatmeal
Grape_Nuts_Flakes
Apple_Jacks
Trix
Total_Corn_Flakes
Cheerios
Triples
Wheat_Chex
Raisin_Squares
Golden_Grahams
Frosted_Mini-Wheats
Multi-Grain_Cheerios
Wheaties
Special_K
Maypo
Nutri-grain_Wheat
Crispix
Honey-comb
Strawberry_Fruit_Wheats
Kix
Rice_Krispies
Corn_Chex
Corn_Flakes
Shredded_Wheat_'n'Bran
Cream_of_Wheat_(Quick)
Shredded_Wheat_spoon_size
Rice_Chex
Shredded_Wheat
Puffed_Wheat



Interpreting PCA

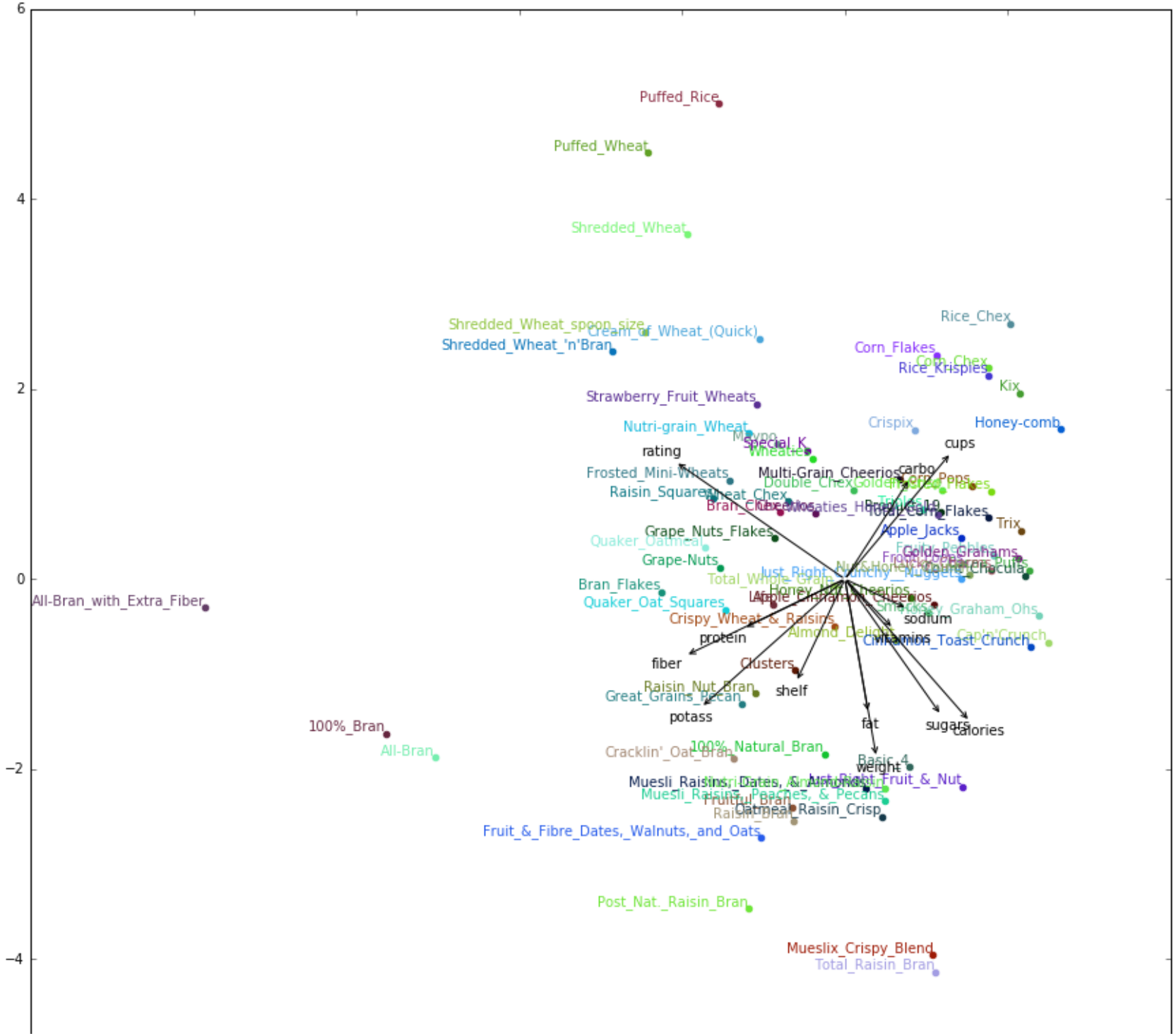
- With the visualization, we can see that PCA does place similar items together
- It also placed different items apart
- But how do we interpret why PCA placed some items here, and some items there?

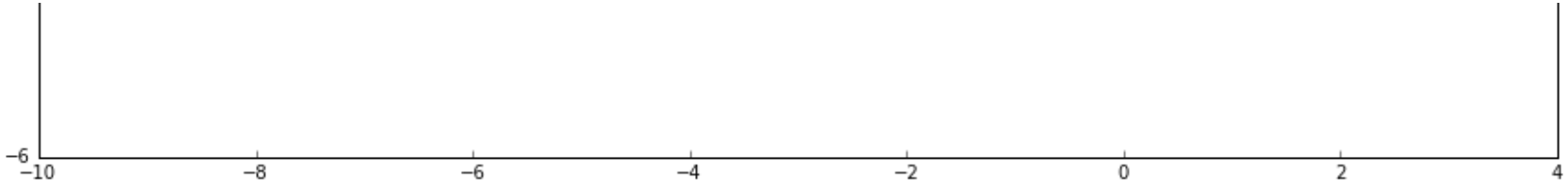
- We can add artificial items, one for each feature
- The item has mean fearure values for all but one feature
- For that feature it has max possible feature value value
- This is the same as adding $\{0, 0, \ldots, 0, v, 0, \ldots, 0\}$ to the standardized data
- Using the cereal example. We set v =max_standardized_feature_values
- The location of a projected feature tells the direction in which that feature is important

In [24]:
```
#generate 77 colors, one for each cereal, then 13 black color for feature
color = ["#%06x" % random.randint(0, 0xAAAAAA) for i in range(0, cereal.shape[0])] \
+ ['black' for i in range(cereal.shape[1])]

cereal2 = StandardScaler(with_std=True).fit_transform(cereal) #standardize
vmax = np.max(cereal2)
# Add artificial rows
cereal2 = np.concatenate([cereal2, vmax*np.identity(cereal2.shape[1])])
# Use feature name as the label
names = np.concatenate([names, cereal.columns])
```

In [25]:
```
pca = PCA(n_components=2) #get PCA
pca.fit(cereal2)
cereal_pca = pca.fit_transform(cereal2)
```

In [26]:
```
#scatter plot
for x, y, c in zip(cereal_pca[:,0], cereal_pca[:,1], color):
    if c is not 'black':
        plt.scatter(x,y,color=c)
#labels
for label, x, y, c in zip(names, cereal_pca[:,0],cereal_pca[:,1], color):
    if c is 'black':
        plt.annotate(label, xy = (0, 0), xytext = (x, y),
                     textcoords = 'data', ha = 'center', va = 'bottom',
                    color="#000000", arrowprops = dict(arrowstyle = '<-', connectionstyle =
'arc3,rad=0'))
    else:
         plt.annotate(label, xy = (x, y), xytext = (-0, 0),
                     textcoords = 'offset points', ha = 'right', va = 'bottom',
                    color=c)
```


6
4
2
0
−2
−4
Puffed_Rice
Puffed_Wheat
Shredded_Wheat
Shredded_Wheat_spoon_size
Cream_of_Wheat_(Quick)
Shredded_Wheat_'n'Bran
Rice_Chex
Corn_Flakes
Corn_Chex
Rice_Krispies
Kix
Strawberry_Fruit_Wheats
Crispix
Honey-comb
Nutri-grain_Wheat
Special_K
Wheaties
cups
rating
carbo
Frosted_Mini-Wheats
Multi-Grain_Cheerios
Raisin_Squares
Wheat_Chex
Trix
Apple_Jacks
Grape_Nuts_Flakes
Quaker_Oatmeal
Golden_Grahams
Grape-Nuts
Chocula
Bran_Flakes
Honey_Nut_Cheerios
All-Bran_with_Extra_Fiber
Quaker_Oat_Squares
Apple_Cinnamon_Cheerios
Crispy_Wheat_&_Raisins
sodium
protein
Cinnamon_Toast_Crunch
fiber
Clusters
Raisin_Nut_Bran
shelf
Great_Grains_Pecan
potass
fat
sugars
calories
100%_Bran
All-Bran
Cracklin'_Oat_Bran
100%_Natural_Bran
Basic_4
weight
Muesli_Raisins,_Dates,_&_Almonds
Just_Right_Fruit_&_Nut
Muesli_Raisins,_Peaches,_&_Pecans
Fruitful_Bran
Oatmeal_Raisin_Crisp
Raisin_Bran
Fruit_&_Fibre_Dates,_Walnuts,_and_Oats
Post_Nat._Raisin_Bran
Mueslix_Crispy_Blend
Total_Raisin_Bran



Reading

- Interesting read about eigenfaces https://en.wikipedia.org/wiki/Eigenface (https://en.wikipedia.org/wiki/Eigenface)

Homework: PCA

- Take around 100 sample digits from the mnist dataset which contains images from hand written digits
- You can do it in python as from sklearn import datasets digits = datasets.load_digits() n = digits.target.shape[0]

```
import random
random.seed(123)
indices = np.array(list(set([random.randint(0, n) for i in range(100)])))
labels = digits.target[indices]
data = digits.data[indices]
```

- If you does not use Python, you can find two csv files in the data.tar.gz (file names digits_*)
- Now apply PCA to reduce the data to 2D
- How much variance can the 2D PCA explain?
- Visualize the 2D data as a scatter plot, and annotate the dots using the digit label.